Weft: Evidence-Gated Version Control for Autonomous Agent Swarms

Pranab Sarkar — Independent Researcher
developer@pranab.co.in · ORCID: 0009-0009-8683-1481
Preprint v1 — August 2026 — License: CC-BY 4.0 — DOI: 10.5281/zenodo.21882499
Code and data: github.com/spranab/weft · weftgate.com

Abstract

Version control systems assume that a human reads each change before it is integrated. Commit messages are prose for a reader; pull requests exist to partition work into human-reviewable units; branch protection encodes "someone approved this." Autonomous coding agents violate that assumption by roughly two orders of magnitude in throughput, leaving practitioners to choose between throttling agents to human reading speed and abandoning pre-integration verification altogether.

I present **Weft**, a version-control and coordination protocol in which integration is gated by *machine-checkable evidence bound to exact content* rather than by human attention. Weft contributes: (i) a change object carrying an explicit **read-set**, enabling detection of semantically stale work whose textual footprint does not conflict — a failure class three-way textual merge cannot observe; (ii) evidence bound to a Merkle **materialization manifest**, so an attestation refers to specific bytes rather than a mutable ref; (iii) capability-based agent identity with authorization frozen at certification time, so credential expiry and revocation cannot retroactively invalidate history; and (iv) a batching-and-bisection gate that amortizes verification across commutable work while isolating faults.

I implement Weft in ~7,000 lines of Rust and evaluate it on four workloads spanning code, structured data, and prose. Across all four, output assembled without a gate fails its own validator, while gated output passes; a 50-agent/100-task workload lands 82 changes in 15 certified landings with 8/8 stale-reads, 8/8 seeded defects, and 3/3 revoked credentials refused. The protocol specification underwent adversarial review by two frontier language models, an executable prototype, continuous integration, and public critique, yielding 92 dispositioned findings — including one fatal content-model defect that fuzzing and model review both missed and only a live demonstration exposed.

1. Introduction

The design center of Git is a human patch-flow: thousands of contributors, email-mediated review, no central authority [1]. Every layer built atop it — pull requests, branch protection, merge queues, code owners — refines the same loop, in which a person inspects a proposed change before it becomes part of the artifact. That loop is not incidental. It is the *only* pre-integration check most repositories have, and human attention is its scarce resource.

Autonomous coding agents break the resource assumption rather than the workflow mechanics. A single agent can produce more reviewable change in an hour than a practitioner carefully reads in a day, and agent counts are multiplicative. Practitioners consequently land on one of two unattractive equilibria:

1. **Preserve the human gate.** Parallel agents serialize behind a single reviewer. Throughput is capped by reading speed regardless of how many agents are deployed.
2. **Abandon the gate.** Changes are skimmed and approved. Nothing verifies anything before integration, and unverified generated output becomes the artifact.

Neither degrades gracefully. The first wastes the capability; the second accumulates unverified state, which is precisely the regime in which model-generated errors — plausible-looking but wrong — are most damaging.

My position is that the dilemma is false, and rests on a historical accident: review became the gate not because humans excel at reading diffs, but because *something* had to check before integration and attention was the only checker available. Most properties that matter about a change are machine-checkable: whether it compiles, whether tests pass, whether a citation resolves, whether a schema holds, whether the author was authorized, and — the property I argue is most neglected — whether the reasoning that produced it is still valid against current state.

Weft makes evidence the integration gate. Humans retain the decisions only they can make (what should be true, and what must be proven), and cease to be the mechanism by which each change is checked.

1.1 Contributions

- A change representation carrying **intent, provenance, footprint, and read-set**, enabling *stale-reasoning detection*: rejecting work whose inputs

changed even when its outputs do not textually conflict (§5.3).

- **Manifest-bound evidence**: attestations reference a Merkle root over materialized content, making "verified" a statement about bytes rather than about a mutable reference (§6.1).
- **Certification-time authorization freezing**: capability validity is evaluated once, at certification, so expiry and revocation gate future integration without invalidating settled history (§4.3).
- A **batching-and-bisection gate** that amortizes one verification run across commutable work and isolates faults in logarithmic rounds (§6.2).
- **Instruction provenance**: repository content is data unless its authors hold an `instruct` capability, converting prompt injection from an ambient hazard into a detectable policy violation (§6.4).
- An implementation and an empirical evaluation across code, data, and prose workloads, with all artifacts and recorded runs published (§8).

---

2. Background and related work

Distributed version control. Git [1] and Mercurial establish content-addressed, tamper-evident history. Weft retains content addressing and distribution and departs on three points: the unit of change, the merge decision, and identity.

Patch-commutation systems. Darcs and Pijul [2] model history as sets of patches with intrinsic dependencies, so independent patches commute. Weft adopts this stance — state is an unordered set — but implements identity via an RGA-family CRDT [3] over line identities rather than a patch-theory calculus, trading algebraic generality for an implementation whose determinism is directly fuzzable.

CRDTs and collaborative editing. RGA and successors provide convergent concurrent text editing [3]. Weft uses CRDT machinery for *placement* convergence only, and explicitly declines the stronger claim: semantic correctness is delegated to evidence. Conflicts are surfaced as first-class, hashable records rather than silently resolved.

Optimistic concurrency control. Weft's read-set is deliberately the read-set of an optimistic transaction [4]. Databases have long validated, at commit time, that a transaction's read-set has not been mutated by a concurrent writer. My observation is that version control never adopted this: a merge validates *writes* against writes, never reads against writes. For human authors the gap

was tolerable because reviewers reconstruct intent; for agents, whose reasoning inputs are explicit and machine-recordable, it is both consequential and cheap to close. Weft applies OCC validation to source integration.

Verifiable logs and supply-chain attestation. Certificate Transparency popularized append-only verifiable logs with equivocation detection; in-toto [5] and SLSA [6] define attestations about how artifacts were produced, and Sigstore provides signing infrastructure. Weft's landing log is such a log, and its evidence objects are attestations in the in-toto sense. The distinction is position in the lifecycle: supply-chain frameworks attest to *builds of already-integrated source*, whereas Weft makes attestation the precondition of integration itself.

Capability systems. UCAN and SPKI/SDSI define attenuated, delegable authority. Radicle [7] demonstrates keypair identity in a peer-to-peer forge. Weft combines both — delegation chains terminating at a genesis authority set — and adds a temporal rule (§4.3) required for an append-only history.

Merge queues and change-centric review. Bors-style merge queues serialize and batch integration, and Gerrit centers review on changes rather than branches. Weft's gate is a merge queue whose admission criterion is signed evidence rather than a status flag, and whose batching exploits commutation rather than optimistic speculation over a linear branch.

To my knowledge, no existing system combines capability-scoped agent identity, read-set validation, and content-bound evidence as *integration* preconditions.

3. Design goals and threat model

Goals. (G1) Integration is gated by machine-checkable evidence about exact content. (G2) Concurrent, non-interfering work integrates without human arbitration. (G3) Every integrated unit carries a verifiable chain from content to authorizing human. (G4) Coordination state replicates with the repository. (G5) Compatibility: existing repositories and existing checks are usable without rewriting.

Adversaries. I assume agents are *not* adversarial in the Byzantine sense but are unreliable in specific, characteristic ways: they fabricate plausible references, introduce syntactically valid but semantically wrong code, and reason from stale context. I additionally assume: a compromised or over-permissioned agent key; repository content authored to manipulate an agent reading it; and a replicating peer that lies about history.

Explicit non-goals. Weft does not attempt to make agents correct, to adjudicate subjective quality without a human-designated judge, or to provide global consensus. Trust is per-repository and rooted in a genesis authority set; disagreement forks.

---

4. The Weft model

4.1 Objects

Twenty object types are content-addressed (BLAKE3), serialized as deterministic CBOR, and Ed25519-signed with domain separation. Canonical bytes *are* the object: relays never re-serialize, and decoders reject non-canonical encodings. Signing covers the pre-signature envelope; the object identifier is the hash of the complete signed envelope.

The core types are `change` (operations plus intent, footprint, read-set, provenance), `patch` (the operations), `state` (a set of changes in delta form), `manifest` (the signed result of materializing a state), `capability`, `evidence`, `policy`, `landing`, and `certificate`.

4.2 Content model

Files and lines carry stable identities of the form `(patch-oid, ordinal)`, assigned at creation and immutable thereafter; editing a line is deletion plus insertion of a new identity. Because operations reference identities rather than positions, a patch means the same thing in any context containing its dependencies — a queued change does not go stale while it waits, and cherry-pick is free.

Materialization is a pure function of the change *set*. Sibling ordering at a shared anchor is total (descending patch identifier, ascending ordinal within a patch), so all nodes derive byte-identical trees. A `manifest` publishes Merkle roots over the tree, the file map, and the conflict set, making materialization a checkable claim rather than a private computation.

Conflicts are classified rather than resolved: concurrent insertions at one anchor are *advisory markers* (deterministic order applies; the state remains clean), while edit-delete, mode, tree, and edit-remove classes are conflicts. This distinction matters in practice: treating same-anchor insertion as a conflict would serialize every append-heavy file, which is exactly the hot path in swarm workloads.

4.3 Capabilities and temporal authorization

An actor is a public key. Authority flows through attenuated delegation chains terminating at a genesis authority key: each link's audience authors the next, each scope is a subset of its parent's, and every link carries an expiry. Unauthorized writes are not rejected by policy — they are unrepresentable, because every object names the capability chain under which it was produced.

A naïve implementation evaluates chains at validation time, which makes history rot: when a six-hour worker credential expires, changes it produced become invalid. Weft evaluates authorization **once, at certification time**, and freezes it. Expiry and revocation therefore gate future integration only. This rule was introduced in response to a review finding (§7.2) and is, I believe, a necessary property of any append-only history whose authorization is delegated and time-bounded.

4.4 Landings

A protected reference advances only through a hash-chained log of `landing` objects, each certified by a quorum of gate keys named in genesis (one key for a solo deployment; a Byzantine quorum for a federation). Certification requires, among other checks: the target state is a superset of the base (history removal is unrepresentable); the manifest matches independent re-materialization; each change's declared footprint matches its patch's actual touched paths; each capability chain is valid; each read-set is fresh; and the policy's evidence requirements are met.

Two certified landings claiming the same slot constitute equivocation. A follower reports the fork and freezes rather than choosing, because a consistency-preserving trunk must not select arbitrarily.

---

5. Concurrency

5.1 Commutation

Because state is a set of identity-anchored changes, work with disjoint footprints is independent by construction. There is no merge step for such work and no conflict to arbitrate — the gate simply admits it together.

5.2 Batching

The gate accumulates proposals, groups those with disjoint footprints, fixes a single target state, and verifies once. In evaluation, 500 disjoint proposals

became one certified landing (§8.2). Because verification cost is dominated by evidence execution — the repository's test suite — this amortization, not raw protocol throughput, is the operative performance property.

5.3 Read-sets and stale reasoning

Consider two agents. Agent A rewrites `api.rs`. Agent B reads `api.rs`, reasons about it, and writes `client.rs`. Under any textual merge, B integrates cleanly: the footprints do not intersect. Yet B's output encodes assumptions about a version of `api.rs` that no longer exists.

Weft changes carry a `reads` field: digests of observed content (optionally region-scoped) and identifiers of observed intents and notes. At certification the gate compares these against the base state and, per policy, rejects or warns. A change's own footprint is excluded from its staleness check, without which append workflows self-invalidate.

I regard this as the clearest instance of this paper's general thesis. The check is trivial once reasoning inputs are recorded; the reason version control never performed it is that human authors' inputs were never legible to the system. Agent inputs are.

---

6. Verification

6.1 Evidence

An `evidence` object binds a pinned recipe (command, environment digest) and its results to a specific `manifest`. Policy names which recipe digests count, how many attestations are required, and from which trust roots. An evidence object counts only if every result passes, the recipe digest is one policy names, and the manifest matches the landing's target.

Three properties follow. Recipes are pinned, so an agent cannot substitute a weaker checker. Evidence is content-bound, so "tests passed" is a statement about specific bytes rather than about a branch at some moment. Attestor independence is measured in *distinct trust roots* rather than distinct keys, since minting keys is free.

A recipe is a command: the gate materializes the candidate state into a scratch directory, executes there, and treats exit status zero as passing. Existing test suites therefore serve as gates unmodified — a property I consider essential to adoption and validate empirically (§8.3).

6.2 Bisection

When a batch fails verification, punishing the batch would make one careless agent block many careful ones. The gate instead splits the batch into cohorts and re-verifies, isolating the offending change in logarithmic rounds while innocent work continues to batch.

6.3 Sandboxing and the position of judgement

Recipes execute in a fresh user and network namespace: no network, no elevated privilege, fresh scratch per run. This has an architectural consequence I did not anticipate but consider correct. Because recipes have no network, a model-based judge *cannot* be a recipe. Judges are therefore independent attestors that sign evidence from outside the gate, which is precisely the role distinct trust roots were designed for. **Deterministic checks run inside the gate; judgement signs from outside it.** For subjective artifacts this separation is not a limitation but the correct decomposition.

6.4 Instruction provenance

Repository text is untrusted model input. Weft labels every retrieved object `instruction` or `data` according to whether its authors hold an `instruct` capability for the enclosing path, and gates refuse to execute recipes whose authoring chain lacks it; mixed-authorship files are always data. I am explicit that this is provenance, not control: a model may still be influenced by hostile text already in its context. The contribution is converting an ambient hazard into a *detectable policy violation with a signed trail*, which is the strongest claim a protocol can truthfully make here.

---

7. Implementation and specification process

7.1 Implementation

The reference implementation is ~7,000 lines of Rust: `weft-core` (canonical CBOR, signed envelopes, materialization and manifests, capability validation, certification), `weftd` (object store, gate and merge queue, governance console), `weft-mcp` (an agent interface over the Model Context Protocol), and `weft-cli` (porcelain and a two-way git bridge). The hub is crash-durable via an append-only log whose replay re-verifies every signature and truncates torn tails; replication is verify-don't-trust, with followers re-deriving the certified chain rather than adopting a peer's claimed head.

7.2 Adversarial specification review

The specification was hardened by five distinct classes of adversary, each of which found defects the others missed. Two frontier language models reviewing independently produced 77 findings and converged on the same fatal flaws — a genesis hash fixed-point, a non-convergent reference update, an undefined policy activation rule, retroactive authorization invalidation — and independently proposed the certified landing log. An executable prototype contributed 9 findings that paper review could not surface, including two cross-object hash cycles. A clean continuous-integration environment exposed a test that asserted one of two equally valid CRDT orderings. Public critique contributed six positioning and roadmap findings.

The most instructive defect, W9, was found only by building a demonstration: the edit-delete conflict rule fired on *sequential* history, because every multi-line insertion forms a chain, so deleting any non-terminal line conflicted with its own successors — making mid-file deletion effectively impossible. A determinism fuzzer had run 300 scenarios without catching it, because it asserted convergence rather than cleanliness. I report this as evidence for a methodological claim: **demonstrations are a distinct verification instrument, not a presentation layer.**

All 92 findings and their dispositions are published.

---

8. Evaluation

All figures are from recorded runs published in the repository; benchmark numbers are from a 32-core Windows workstation with an in-memory store.

8.1 Does gating change the artifact?

I ran four workloads twice with *identical* agents and contributions: once merged in arrival order (the naive pipeline) and once through the gate. Each workload contains one flawed contribution modelled on a characteristic agent failure. The same validator was then executed over both outputs.

Workload	Flaw	Ungated	Gated
CSV report, 4 agents	format drift, non-numeric cell	**fail**	**pass**
Python module, 3 agents	unbalanced parenthesis; module will not import	**fail**	**pass**
Policy brief, 3 agents	fabricated citation `[7]`	**fail**	**pass**
Existing test suite, 3 agents	function redefined, existing test breaks	**fail**	**pass**

4/4 ungated outputs failed their own validator; 4/4 gated outputs passed.
The agents were identical in both columns; the only variable is whether anything checked before the work became the artifact.

8.2 Swarm behaviour

A 50-agent, 100-task workload on a single repository, deterministic across runs: 82 changes landed across 15 certified landings, with a maximum of 40 independent changes in a single landing. All 8 stale-read changes were rejected, all 8 seeded defects were isolated by bisection, and all 3 post-revocation attempts were refused at certification. Notably, the stale-read rejections involved changes with non-overlapping footprints — the class textual merge cannot detect.

8.3 Adoption cost

For the fourth workload, the gate's evidence recipe was the repository's own `python -m pytest -q`, unmodified. No verification logic was written for Weft.

8.4 Protocol overhead

Object ingest (canonical decode, signature verification, store): ~25,000 objects/s. Materialization of a 5,000-change set: ~9 ms. Gate throughput with disjoint work: ~39,000 changes/s, batching 500 proposals into one landing. Fully contended (all work on one file): ~1,000 changes/s, i.e. approximately 1 ms per fully-serialized certified landing.

The engine is not the bottleneck at any plausible agent scale. Real throughput is bounded by evidence execution, which is why batching — not protocol optimization — is the operative design lever.

8.5 Interoperability

An existing repository was imported through the gate, extended by three agents, and exported as conventional git commits carrying provenance trailers, chained onto the original base commit and byte-deterministic across re-exports.

---

9. Limitations

I state these plainly because the alternative is discovered by users.

Single-gate quorum in practice. Multi-gate thresholds are specified and certificate structure supports them, but the reference gate signs alone. Byzantine-tolerant deployments are untested.

Pull-based replication. Followers poll over HTTP; the QUIC frames and push subscriptions in the specification are unimplemented.

Evidence is only as good as the check. A gate that passes everything teaches its users that gating is theatre. Weft enforces *that* a check ran on *these* bytes; it cannot judge whether the check was worth running.

Who verifies the verifier. An agent that writes a defect and a test that accepts it defeats a naive policy. Pinned recipes, distinct trust roots, and human approvals mitigate; heterogeneous evidence quorums are future work. I do not claim this is solved.

Bulk rewrites are the content model's worst case. Formatters, code generators, and lockfiles replace most line identities at once, destroying continuity. Mitigations exist (blob mode, merge drivers, dedicated intents); identity compaction is future work.

Operational surface. Key custody, hub operation, and policy design are real work. Weft makes authority explicit rather than eliminating it.

Evaluation scope. Flawed contributions in §8.1 are planted for determinism and reviewability. They are modelled on observed agent failure modes, but this is a controlled demonstration, not a field study. I have not evaluated Weft on a large production repository under sustained multi-agent load, and that remains the most important missing evidence.

---

10. Future work

Compositional evidence — proofs that survive footprint-disjoint deltas — is the highest-value item, since it removes the remaining throughput ceiling. Heterogeneous evidence quorums combining compilers, property checks, runtime traces, and independent model judges from distinct roots address §9's central open problem. Identity compaction (epochs) bounds tombstone growth and mitigates the formatter case. Beyond these: multi-gate quorums, push-based gossip, object-level encryption for untrusted hosts, and hardware-rooted runner attestation.

11. Conclusion

Integration gating by human attention was a reasonable engineering decision when attention was the only available checker. It is now the binding constraint on autonomous software development, and the common workaround — retaining the ceremony while abandoning the check — is worse than either honest alternative.

Weft demonstrates that the gate can be evidence instead: signed, bound to exact content, evaluated against declared policy, and cheap enough that existing test suites serve unmodified. The empirical result is narrow but unambiguous — across four workloads, identical agents produced failing artifacts without a gate and passing artifacts with one. The design contribution I consider most durable is the read-set: once an agent's reasoning inputs are recorded, validating them is routine, and a class of error that textual merge cannot perceive becomes mechanically detectable.

Humans do not disappear from this picture. They decide what should be true and what must be proven. They stop being the mechanism by which each change is checked.

References

[1] L. Torvalds and J. Hamano. *Git: fast version control system*, 2005.
[2] P. Meunier et al. *Pijul: a distributed version control system based on patch theory*.
[3] H.-G. Roh, M. Jeon, J.-S. Kim, J. Lee. "Replicated abstract data types: building blocks for collaborative applications." *JPDC*, 2011.
[4] H. T. Kung and J. T. Robinson. "On optimistic methods for concurrency control." *ACM TODS*, 6(2), 1981.
[5] S. Torres-Arias et al. "in-toto: providing farm-to-table guarantees for bits and bytes." *USENIX*

Security, 2019.
[6] OpenSSF. *SLSA: Supply-chain Levels for Software Artifacts*.
[7] Radicle. *A peer-to-peer collaboration network*.
[8] B. Laurie. “Certificate transparency.” *ACM Queue*, 12(8), 2014.
[9] UCAN Working Group. *User-Controlled Authorization Networks*.

Artifact availability. Specification (RFC-0001), implementation, all recorded runs, the complete review log with 92 dispositioned findings, and the showcase artifacts of §8.1 are available at github.com/spranab/weft. A live read-only instance runs at demo.weftgate.com.